In one embodiment of the present invention, the structure templates are stored in the DIT itself. This simplifies use of the templates by eliminating the need for each user to parse a template file. FIG. **2** shows a sub-tree **200** of an exemplary DIT into which structure templates have been loaded. The *Organization* node **202**, with the node name "o=ParentCorp" has, as its children, a set of structure template nodes **204**, **206**, **208**, **210**, and **212**, which correspond to entries in Table **1**. Node **202** also has as its children two other *Organization* nodes **220** and **240**. Node **220**, in turn, has as its children a *People* container node **222**, a second set of structure templates **224**, **226**, and **228**, and a *Groups* container node **230**, which has child nodes **232** and **234**. Likewise, node **240** has as its children a set of structure templates **242**, **244**, **246**, and **248**, a *People* container node **250**, and a *Groups* container node **260**, which, in turn, has as its children two nodes **262** and **264**.

In one embodiment, the structure templates are stored in a linear fashion (e.g., not in a hierarchical fashion). That is, all the nodes corresponding to entries in the same structure template file are children of the same parent node, and all structure template nodes are leaf nodes (i.e., nodes without children). The present invention, however, is not limited to such a storage scheme.

In one embodiment of the present invention, the structure template entries are defined in a template file, such as the XML file appearing in Table 1, which is then read and parsed by a software program, which in turn creates nodes in the DIT that correspond to the template entries in the file. In addition, an administrator may choose to create some template nodes in the tree manually. Alternatively, the template entries may be read directly from the template file by each user. This approach, however, may be less efficient.

With the structure template nodes in place, a software program can be created to modify the directory using the structure templates. Extracts from Java code used in implementing one such program appears in Table **2**, which defines a Java class called *PersistentObject*.

When creating a new node in the DIT, a user checks the structure templates to ensure that proper tree structure is maintained. To this end, it first locates the proper set of structure templates. The purpose of allowing the DIT to contain multiple sets of structure templates is to facilitate different structural guidelines throughout the directory. In the system **200** of FIG. **2**, for instance, the *Organization* node **202** has two *Organization* child nodes **220** and **240**,

each of which have a set of structure template nodes as children. Thus, the two sub-trees (i.e., the sub-tree under node **220** and the sub-tree under node **240**) are controlled by two different sets of structure templates, each set dictating a potentially different structure.

In one embodiment, locating the proper set of structure templates amounts to locating the closest *Organization* node with a set of templates as one traverses up the tree. Thus, a user adding a new node must traverse up the line of ancestry (i.e., to the parent of the current node, then to the parent of that parent node, and so on) until it finds a node with structure templates as its children.

The *PersistentObject* class contains a method called *addChild* (Table **2**), which is used to add a child node to an existing node of the directory tree. The existing node is the instance of *PersistentObject* class upon which the call to *addChild* is made. The *addChild* method takes a single argument of type *PersistentObject*, which is the child node to be added to the tree, under the aforementioned existing node.

Another method is used to retrieve structure template information. In one embodiment, this method is provided with, as one of its parameters, the location in the DIT where the new node is being created. Given this information, the method starts with the parent node of the new node, traversing up the line of ancestry--all the way to the root node, if necessary--until a usable set of structure templates is located. At each node along the way, the method checks for any child structure template nodes with a "class" attribute value matching the Java type of the newly created object. If any exist, they are immediately returned, ending the search. Otherwise, the search continues, moving up the tree one level at a time

Once the Structure Template information is returned, an Entity Manager class creates the necessary child entries, if any, as defined by that information. In case multiple template entries are returned, of course, the proper template may be chosen for use. To this end, the "filter" and, if necessary, "priority" attributes are used. Once the proper template is picked out, an iterative child-creation process comprising four steps may be used to create all the necessary nodes. First, the values for the "childNode" attribute in the structure template are extracted. Next, each such value, along with a unique identifier for its parent node (the two values hereafter referred to as one "record") , is pushed onto a stack (i.e., a list data structure, wherein stored records are retrieved in the opposite order to that in which they are stored).

Third, one record is popped off the stack, and the search described above is repeated to find the structure template entry information for that record. Finally, the child node itself is created in the directory tree. These four steps are repeated until the stack is empty. At that point, the necessary sub-structure for the new node has been created.

The present invention has many advantages over existing systems. The system of the present invention can be used to automatically enforce a standard structural hierarchy for complex directory objects. Moreover, it allows software applications to define complex composite objects in terms of other complex composite objects and directory entries. Also, the structure templates, which can be created and modified by user-friendly graphical user interfaces, allow the system administrator to easily set up and customize different structures throughout the DIT. Yet another advantage is realized in that the structure information is maintained in one place. This simplifies the task of describing and modifying directory structure. The system also provides meta information about the relative location of directory objects. The template plates are extensible, meaning their format may be modified to provide more information than described herein. For example, aci information can be added, so that default acis are oriented when each entry is created. This customization may go into Creation Template

The preferred and alternative embodiments described above and illustrated in the attached drawings and tables have been provided by way of illustration only, and numerous other embodiments of the disclosed invention will be apparent to those skilled in the arts from consideration of the above description and attached drawings. Accordingly, limitations on the subject invention are to be found only in the claims set forth below.